STA 32 R Handout 6. Generating Random Variable

1 Generate Random Variables

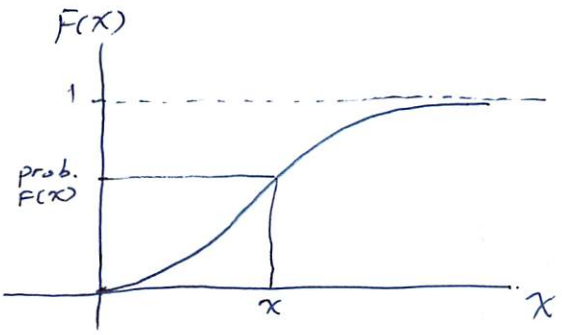
We have seen in R how to generate binomial and poisson random variables. Suppose now you are working with a distribution that R doesn't have a build-in function for, and you need to generate values coming from this distribution. Nice enough, there are many algorithms out there to do this. We will learn the simplest methods of all: the inverse CDF method.

1.1 Continuous Distribution

We will first look at the easier case: when the random variable has a continuous distribution. Recall that for any continuous r.v., we can write down the p.d.f and also the CDF. Let $F(X)$ denote the CDF of the random variable X. Suppose I have samples $x_1, x_2, ..., x_n$, which all come from the same distribution as X, then one can prove that $F(x_i) \sim U(0, 1)$, i.e. the values $F(x_i)$ are uniformly distributed between 0 and 1. Using this property, if we are giving a probability and a CDF, we can recover the original value by the following operation:

$$x_i = F^{-1}(prob)$$

and this is where the name 'inverse CDF' comes from.



Example 1 *Suppose X takes on value between 0 and 1, and has a pdf $f(x) = 3x^2$. Then by integrating, we get $F(x) = x^3$. The inverse is $F^{-1}(x) = x^{1/3}$. If prob = 0.6, then we know that the x value that gives a cumulative probability of 0.6 is $x = F^{-1}(0.6) = 0.8434$.* To generate values from this distribution in R, we can do the following:

```
F inv = function(p){
x = p^(1/3)
return(x)
}
p = runif(1, 0, 1)  #This command is used to generate 1 uniformly distributed
#value between 0 and 1
x value = F inv(p)
```

Note that since the C.D.F. is an increasing function (and nondecreasing in rare cases), the inverse exists and can be calculated. Numerical approximation can be done if the analytic form cannot be found.

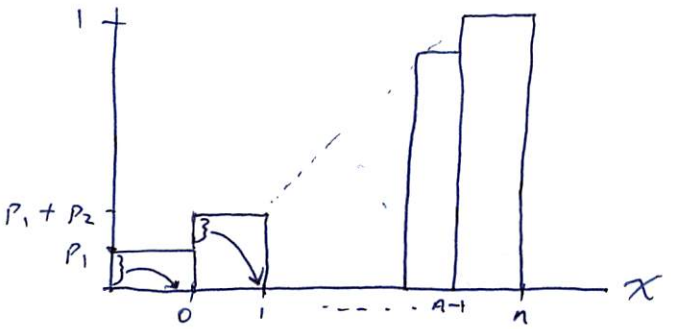
1.2 Discrete Distribution

The discrete case is a bit complicated, since our CDF is not continuous anymore. To generate from a discrete distribution, consider a discrete r.v. X with p.m.f.

$$P(X = x_i) = p_i, \qquad i = 1, \ldots, n, \sum_i p_i = 1$$

So this r.v. can take on n values. Generate a uniformly distributed r.v. U and set

$$x = \begin{cases} x_1, & \text{if } U < p_1 \\ x_2, & \text{if } p_1 \le U < p_1 + p_2 \\ \vdots & \\ x_j, & \text{if } \sum_{i=1}^{j-1} p_i \le U < \sum_{i=1}^{j} p_j \\ \vdots & \\ x_n, & \text{if } \sum_{i=1}^{n-1} p_j \le U < 1 \end{cases}$$



The pseudo-code for this algorithm is:

1. Generate a uniformly distributed random number U
2. - If $U < p_1$, set $x = x_1$ and STOP. Else
   - If $U < p_1 + p_2$, set $x = x_2$ and STOp. Else
   - ⋮

Example 2 *Suppose we want to simulate from the discrete distribution with*

$$p_1 = 0.20, p_2 = 0.25, p_3 = 0.40, p_4 = 0.15$$

We can do the following:

Generate a random number U.

If $U < 0.2$, set $x = 1$ and STOP.

If $U < 0.45$, set $x = 2$ and STOP.

If $U < 0.85$, set $x = 3$ and STOP.

Otherwise, set $x = 4$.

To do this in R, we can use the following code:

```
# function to simulate from a given discrete distribution
# required inputs  x=values the distr  takes, probabilities  number to generate
simdiscrete <- function(x, probs, n gen){
xprobs <- data frame(x=x,probs=probs)
# sort the data in order of x, with cumulative probabilities
xprobs sorted <- xprobs[order(xprobs$x), ]
cum probs <- cumsum(xprobs sorted[,2])
xprobs sorted <- cbind(xprobs sorted, cum probs)
# generate and loop
urandom <- runif(n gen, 0, 1)
sim vector <- rep(0, n gen)
for(i in 1 n gen) sim vector[i] <- min(xprobs sorted[ 1][which(cum probs > urandom[i])])
# output
return(sim vector)
}
x = c(1, 2, 3, 4)
prob = c(0 2, 0 25, 0 4, 0 15)
sim data = simdiscrete(x, prob  1000)
hist(sim data)
```